título

Procesamiento masivo de datos mediante Cassandra y Spark

Máster en Sistemas Informáticos Avanzados
Septiembre de 2016

Autor:

Xabier Zabala Barandiaran

Supervisores:

German Rigau i Claramunt
UPV/EHU
Iñigo Etxabe y Beñat Aranburu
Datik Información Inteligente S.L.

Agradecimientos

En primer lugar, quisiera expresar mi gratitud a las personas que han posibilitado la concepción y el desarrollo de la presente tesina. Agradezco a German Rigau i Claramunt, director del proyecto en la UPV/EHU, el asesoramiento ofrecido durante el transcurso del mismo. Doy también las gracias a Iñigo Etxabe y Beñat Aranburu, supervisores en Datik Información Inteligente, por poner a mi disposición todos los medios tecnológicos necesarios para llevar a cabo el trabajo y por el trato ofrecido desde el primer día.

Agradecer, cómo no, a mis padres José Javier Zabala y María Pilar Barandiaran el esfuerzo desempeñado para facilitarme, en la medida que les ha sido posible, el camino que he recorrido hasta el día de hoy. Me congratula haber sabido responder satisfactoriamente a la confianza que siempre han depositado en mí. Por todo lo dicho y por todo lo que suponen para mí, un beso enorme para los dos.

No me quisiera olvidar de aquellas personas que me han acompañado durante este maravilloso periplo. Doy las gracias a los amigos de toda la vida por el apoyo incondicional mostrado desde la distancia y a la gente que he tenido la fortuna de conocer durante los años de universidad. Quisiera agradecer especialmente a las personas que en este periodo se han ganado a pulso el privilegio a ser parte importante de lo que me resta de existencia, a los cuales me atrevo a mencionar aún con el temor de dejarme a alguna en el tintero: Adrián Núñez, Eider Irigoyen, Jaime Altuna, Marta García y Mikel Etxeberria. No hallo palabras para expresar todo lo que os debo.

Por último, pero no por ello menos importante, quisiera evocar a todos los docentes que han tomado parte en mi formación desde aquel Septiembre del 2008 y agradecer a todos ellos el conocimiento compartido y el esfuerzo invertido en mí durante estos años.

Gracias de todo corazón a la gente mencionada en este breve capítulo por haber hecho de mí un mejor profesional y una mejor persona, además de darme las fuerzas necesarias para seguir evolucionando en ambos aspectos de cara al futuro.

Resumen

Proyecto Final del Máster en Sistemas Informáticos Avanzados. Estudio empírico sobre el rendimiento ofrecido por varias tecnologías emergentes en el campo del Big Data en comparación a una base de datos tradicional a la hora de operar en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

Para llevar a cabo el experimento, dos entornos de prueba totalmente aislados han sido erigidos sobre la misma máquina física. En el primero, se ha construido un clúster compuesto por tres nodos virtuales que operan sobre una misma red privada. Dichos nodos han sido dotados de tecnología necesaria para el correcto funcionamiento de Apache Cassandra [4] y Apache Spark [12]. En el segundo entorno, se ha instalado MySQL Server sobre un nodo virtual de potencia equivalente al clúster mencionado con anterioridad. Una vez habiendo poblado las bases de datos mediante un data-set público de aproximadamente 25GB y diseñadas unas consultas acordes a la naturaleza de los datos, se han ejecutado dichas consultas para así cuantificar el tiempo de respuesta en cada escenario.

El estudio evidencia que a la hora de trabajar con volúmenes masivos de datos el binomio entre Apache Cassandra y Apache Spark mejora sustancialmente los tiempos de procesamiento obtenidos con MySQL, además de ofrecer una solución totalmente escalable y tolerante a fallos. No obstante, para gozar de dichas ventajas se antoja necesario un análisis previo de los datos que se desean tratar, aspecto en el que MySQL demuestra ofrecer una mayor libertad.

Palabras Clave: Apache Cassandra, Apache Spark, Big Data, Clustering, Comparativa, NoSQL.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Desde Aristóteles y su libro Segundos Analíticos [1] hasta Galileo, padre de la ciencia moderna, adalides del conocimiento han proclamado que un método de investigación basado en lo empírico y en la medición, sujeto a los principios específicos de las pruebas de razonamiento es el camino para conocer la verdad.

Hoy en día, época en la que los avances tecnológicos han posibilitado observar y medir de forma exhaustiva un gran abanico de fenómenos, la ingente cantidad de datos que se genera en el proceso es, a veces, intratable mediante las tecnologías convencionales, y por ende, es imposible extraer todo el conocimiento que atesoran. El problema, lejos de atenuarse, se acrecienta con el paso del tiempo. Estudios como el realizado por McKinsey Global Institute [2] estiman que el volumen de datos que se genera crece un 40 % cada año y auguran que entre 2009 y 2020 se verá multiplicado por 44 [6].

Para lidiar con dicha problemática, en los últimos años ha irrumpido la necesidad de desarrollar nuevas metodologías y tecnologías que permitan operar eficientemente sobre masas colosales de datos, dando como resultado el nacimiento del Big Data [5].

Empresas de renombre mundial, conscientes de los beneficios que el Big Data les puede reportar en diferentes facetas de su modelo de negocio, ya se han interesado en este fenómeno. De un estudio realizado entre los altos ejecutivos de las firmas que lideran el Wall Street se desprende que el 96 % tiene planeadas ciertas iniciativas relacionadas con el Big Data, y el 80 % ya tiene finalizada alguna [10].

[1] Órganon II de Aristóteles: Recopilación de obras Aristotélicas que incluye el libro Segundo Analíticos
[2] McKinsey Global Institute

1.1. Contexto

Datik Información Inteligente [3] es una empresa tecnológica perteneciente al Grupo Irizar [4] que desarrolla soluciones ITS destinadas a la gestión del trasporte, tanto ferroviario como por carretera y movilidad ciudadana.

Uno de los productos estrella de la entidad es el denominado iPanel, concentrador de información que ofrece al operador de transporte servicios de valor añadido en la gestión de la información generada por su flota. El funcionamiento del servicio se puede resumir mediante la Figura 4:

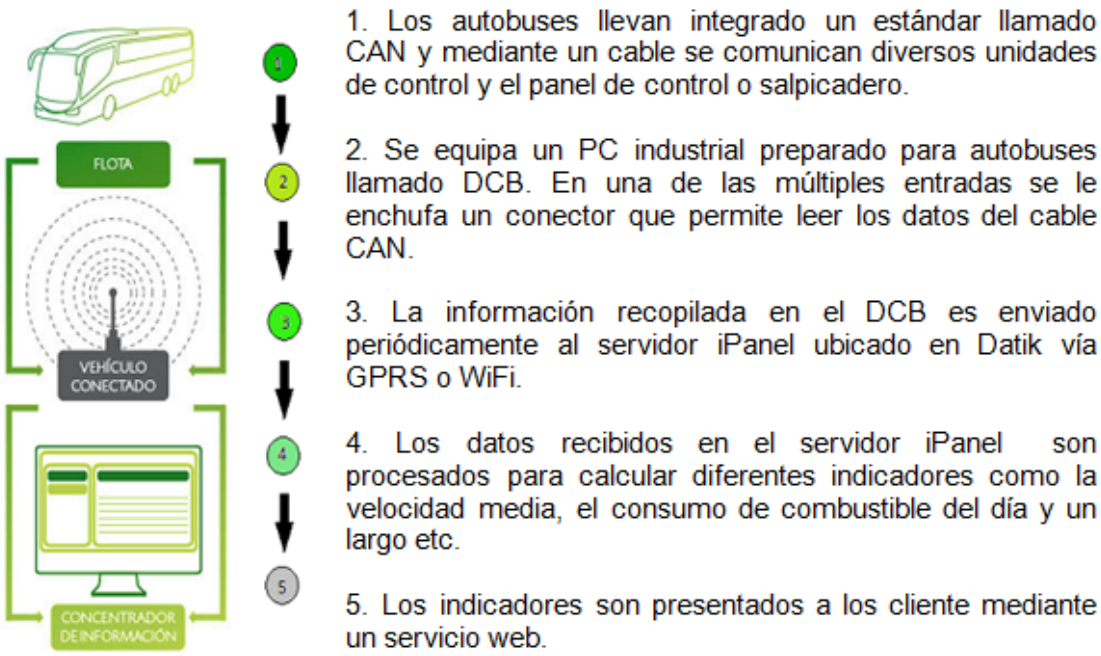

Figura 1: Funcionamiento resumido de iPanel

La incesante integración de nuevos vehículos en iPanel ha generado un crecimiento exponencial del número de registros almacenados en ciertas tablas MySQL. Aunque el volumen actual no llega a suponer riesgo alguno para el funcionamiento del servicio, Datik tiene identificados varios escenarios en los que la situación se podría revertir, causando graves inconvenientes.

[3] http://www.datik.es/
[4] http://www.irizar.com/irizar/

Uno de los problemas, intrínseco a operar sobre una base de datos centralizada, es el denominado punto único de fallo o SPOF. Al tratarse del punto en el cual todos los procesos confluyen, el bloqueo o la caída causada por uno puede afectar al resto. Para mitigar los riesgos que ello conlleva, Datik ha optado por migrar su arquitectura monolítica a una basada en Microservicios [7], logrando así aislar los procesos que conforman sus servicios y mejorar ostensiblemente la estabilidad del sistema. No obstante, dicho cambio no soluciona los problemas inherentes a un proceso, que en caso de caída puede llegar a acarrear un funcionamiento inesperado de otros.

Fiel reflejo de ello es el Cálculo de Indicadores: proceso de ejecución diaria que realiza operaciones aritméticas intensivas sobre datos almacenados en diversas tablas para después, agrupar los resultados en base a diferentes criterios. Siendo dichas tablas las que mayor crecimiento experimentan, el aumento del volumen de las mismas incrementa de forma desorbitada el tiempo necesario para finalizar el cálculo, pudiendo, en un futuro, llegar a tardar más de 24 horas y cancelar los indicadores que ofrecen información del último día.

El objetivo del presente proyecto es proponer soluciones tecnológicas que solvente permanentemente el problema del proceso Cálculo de Indicadores y que a su vez, pueda valer para lidiar con otros obstáculos de la misma índole que pudieran emerger en un futuro.

1.2. Propuesta

Tal y como se ha mencionado en apartados anteriores, la problemática que envuelve al Cálculo de Indicadores es originado por el incremento exponencial de datos que dicho proceso ha de tratar. No sería descabellado pensar que la solución podría pasar por escalar verticalmente la máquina y afinar la configuración de MySQL. No obstante, ambas mejoras son limitadas, mientras que el volumen de los datos seguirá en aumento de forma inexorable, volviendo, tarde o temprano, a tener que lidiar con los mismos problemas del principio.

Siendo imposible reconducir la situación mediante las tecnologías tradicionales, en el presente proyecto se propone realizar un cambio de paradigma que implica migrar las tablas relacionadas con los Indicadores a un modelo distribuido que posibilite escalar la infraestructura horizontalmente. A su vez, se sugiere dividir el problema en dos apartados, almacenamiento y procesado, dotando la infraestructura de tecnología adecuada para ofrecer una respuesta eficaz a cada una de las partes.

En cuanto a almacenamiento se refiere, teniendo en cuenta que los datos a tratar no presentan relación alguna con el resto de las entidades, se propone utilizar una base de datos no relacional. Dentro de la extensa gama de sistemas de almacenamiento NoSQL [5] existentes hoy en día, se ha considerado que Apache Cassandra [4] es el idóneo para solventar el problema gracias a que ofrece una disponibilidad total y un ratio de escrituras por segundo sustancialmente superior en comparación a sus homólogos [8].

Por su parte, para trabajar con el apartado del procesamiento se apuesta por Apache Spark [12]. Se trata de una plataforma de computación en clúster que ofrece una interfaz para programar operaciones paralelizadas y tolerantes a fallo que ha irrumpido con fuerza en el último lustro. A logrado desbancar tecnologías semejantes como MapReduce [2] gracias a la capacidad de almacenar en memoria los resultados intermedios del cómputo posibilitando así acelerar el procesamiento hasta 100 veces [11] en determinados escenarios.

Para cuantificar los beneficios aportados por las tecnologías propuestas, teniendo en cuenta que Datik actualmente almacena una cantidad limitada de datos y que además estos se encuentran sujetos a una clausula de confidencialidad, se ha decidido hacer uso de un Dataset público de aproximadamente 25 GB que recoge la información de los trayectos realizados por los taxis amarillos de Nueva York durante el año 2015. [6]

1.3. Organización del documento

La presente memoria recoge el estudio empírico realizado sobre el rendimiento ofrecido por varias tecnologías emergentes en el campo del Big Data en comparación a una base de datos tradicional a la hora de operar en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

En este primer capítulo se ha introducido la problemática que ha impulsado la creación del presente proyecto y se ha expuesto la propuesta para dar solución a la misma.

En el capítulo 2 se desgrana el funcionamiento de Apache Cassandra y Apache Spark, haciendo especial hincapié en aquellos aspectos que denotan una mejora sustancial en comparación a la arquitectura centralizada.

Una vez en el capítulo 3 se describen los entornos de prueba confeccionados para cuantificar la mejora

[5] `http://nosql-database.org/`
[6] `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`

explica la gestión llevada a cabo durante el proyecto. Se presentan las metodologías utilizadas: Metodologías Ágiles e InterMod (adaptada a las necesidades de este proyecto). A continuación se detallan cada una de las iteraciones llevadas a cabo (como parte de la metodología InterMod): duración, objetivos y tareas realizadas. Al final del capítulo se muestra la documentación asociada a las iteraciones y los objetivos, además del seguimiento de tiempo realizado.

A continuación, en el capítulo 4 se detalla el análisis de requisitos. Primero se detallan los requisitos no-funcionales y luego los funcionales (prototipos en papel llevados a cabo durante las primeras iteraciones que dan una visión global del proyecto).

En el capítulo 5 se explica el diseño e implementación llevados a cabo. Se comienza mostrando la estructura de documentos del proyecto, luego el diseño realizado en base al análisis de requisitos del capítulo 4 y finalmente una visión general de la implementación de la lógica de negocio.

Para finalizar, en el capítulo 6 se presentan las conclusiones, líneas futuras para el proyecto y las lecciones aprendidas.

Fuera de la estructura general de la memoria, tenemos la bibliografia y los apéndices. En estos últimos tenemos las actas de reuniones, las actas de pruebas y la vista de relaciones de la base de datos (de la parte utilizada o creada específicamente para el proyecto).

Capítulo 2

Análisis de las tecnologías propuestas

2.1. Apache Cassandra

Apache Cassandra es una base de datos distribuida que permite operar sobre grandes volúmenes de datos del tipo clave/valor. Se caracteriza por ofrecer una disponibilidad total y mayor escalabilidad lineal en comparación a otras bases de datos NoSQL, utilizando, para ello, una serie de nodos homogéneos que se comunican mediante un protocolo P2P de replicación asincrónica, lo cual permite realizar operaciones de baja latencia para todos los clientes sin necesidad de un servidor maestro.

Fue concebida en el 2008 por los ingenieros de Facebook basándose en sistemas de almacenamientos distribuidos como Dynamo (Amazon) [3] y BigTable (Google) [1] con el propósito de mejorar la funcionalidad de búsqueda en la bandeja de entrada. En 2012, investigadores de la Universidad de Toronto que estudian los sistemas NoSQL concluyeron que .[En] términos de escalabilidad, hay un claro ganador a través de nuestros experimentos. Cassandra logra el mejor rendimiento para el número máximo de nodos en todos los experimentos"[8]. Hoy en día, el mayor impulsor del proyecto es DataStax [1], empresa que operando bajo la licencia de Apache [2] desarrolla tanto ésta base de datos como una infinidad de herramientas que facilitan el uso de la misma.

Actualmente Apache Cassandra es utilizada por infinidad de aplicaciones en negocios modernos, siendo la base de datos elegida por un tercio de las compañías que conforman la Fortune 100 [3]. Claro ejemplo de ello son empresas mundialmente conocidas como

[1] http://www.datastax.com/
[2] http://www.apache.org/
[3] http://fortune.com/2015/04/14/datastax-hp-sales-partnership/

Apple, Facebook o NetFlix, los cuales utilizan Cassandra como parte de su entramado tecnológico desde hace ya unos años. Cabe destacar que el uso del mismo no se limita al mundo empresarial. Muestra de ello es la acogida que ha tenido en el ámbito de la investigación, formando parte en experimentos punteros a nivel mundial como algunos de los realizados en el CERN [9].

Es indispensable tener en cuenta que no se trata de una base de datos de propósito general, por lo que, es de vital importancia conocer cuando se va a poder exprimir su potencial al máximo y cuando no.

Será **recomendable** utilizar Cassandra si:

- se desea que la configuración, mantenimiento y el código sea sencillo.
- se necesitan velocidades muy altas en lecturas y escrituras aleatorias.
- no se necesitan múltiples índices secundarios
- existe alta flexibilidad en la estructura de los datos
- se busca una escalabilidad masiva
- se busca alta disponibilidad

No será **recomendable** utilizar Cassandra si:

- se manejan datos relacionales
- hay transacciones de por medio
- se requiere autorización para acceder a datos
- se necesita latencia baja

2.1.1. Funcionamiento

Debido al modelo distribuido sobre el cual esta desarrollado, el primer paso para comprender el funcionamiento de Apache Cassandra es familiarizarse con los conceptos básicos de los sistemas distribuidos.

El Teorema de Brewer[], también conocido como Teorema CAP , enuncia que es imposible para un sistema de cómputo distribuido garantizar simultáneamente las tres propiedades que se presentan a continuación, solo pudiendo cumplir dos de ellas al mismo tiempo, y acabar cumpliendo el restante tarde o temprano.

- **Consistencia**(Consistency): Todos los nodos ven la misma información al mismo tiempo.
- **Disponibilidad**(Availability): La garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente.
- **Tolerancia al Particionado**(Partition Tolerance): El sistema sigue funcionando a pesar de que haya sido partido por un fallo de red.

Para una base de datos distribuida que promete la disponibilidad completa, como lo es Cassandra, el Teorema de Brewer implica la imposibilidad de garantizar la consistencia total de los datos y la necesidad de esperar un tiempo indeterminado para que las réplicas de un registro modificado se actualicen correctamente.

Dicha actualización corre a cargo de un protocolo Gossip(), encargado del funcionamiento coordinado de la infraestructura. Mediante paso de mensajes periódicos da a conocer el estado de un nodo al resto de sus vecinos, posibilitando de esa manera, que cada nodo pueda mantener actualizadas sus réplicas, además de conocer en cada instante qué nodo esta caído y cual no.

No obstante, existen formas de minimizar el tiempo que hace falta para alcanzar un estado consistente. La primera, es proveer la infraestructura de medios físicos necesarios para acelerar el intercambio de datos entre los nodos y evitar que la red se congestione en el proceso. La segunda, trata sobre la posibilidad que Cassandra ofrece de elegir el nivel de consistencia() con el que se ejecuta cada consulta.

Analizando las características propias de Cassandra y en concreto su estructura, la Keyspace es un espacio de nombres para un conjunto de ColumFamily. Por lo general, se utiliza uno por aplicación y es considerado como el equivalente a una base de datos del modelo relacional. El ColumFamily, a su vez, es capaz de almacenar diferentes columnas, siendo el homólogo de una tabla del modelo relacional. Para finalizar, una columna está compuesta por clave y valor, además de un campo del tipo timestamp gracias al cual se actualizan las réplicas obsoletas.

A la hora de definir un keyspace dos atributos estrechamente relacionados han de ser especificados: Replication Factor y Replica Placement Strategy(). El primero, consiste en un número que indica cuantas copias de un mismo registro se han de almacenar en la infraestructura. Cada nodo posee una réplica para un cierto rango de datos y si uno de ellos falla, otro que posea dicha réplica puede responder a la petición sin tener que interrumpir el servicio. El segundo, define cómo se han de repartir los registros replicados por el anillo, ofreciendo distintas opciones dependiendo si los nodos del clúster se en-

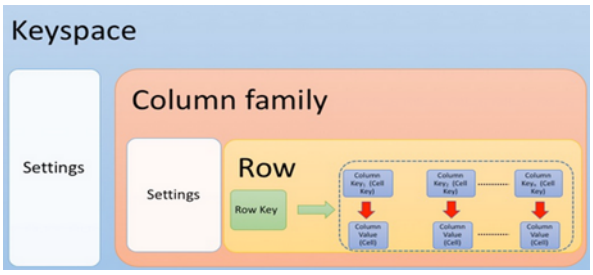

Figura 2: Estructura de Cassandra

cuentran alojados en un único datacenter o repartidos entre varios.

El atributo Replica Placement Strategy cobra especial interés al operar sobre un clúster cuyos nodos están distribuidos en punto geográficos lejanos, ya que permite almacenar cada réplica en diferentes puntos del globo. De esa manera, se consigue minimizar la latencia de las peticiones recibidas desde diferentes puntos del plantea y además, proteger la base de datos de una posible caída a nivel de datacenter debido a diversos catástrofes.

Si un componente de Cassandra tiene especial importancia en su funcionamiento es el ColumFamily. En una base de datos relacional como MySQL, las tablas son diseñadas con el objetivo de minimizar la redundancia de los datos almacenados en ella, siguiendo para lograr dicho objetivo un proceso de normalización[ ]. En cambio, al operar con Cassandra, ocurre exactamente lo contrario. Las tablas se construyen buscando una respuesta rápida a las consultas sin importar que ello implique tener que almacenar datos redundante en la base de datos.

patition key etc
consecuencia: restriccion en las consultas que se pueden realizar

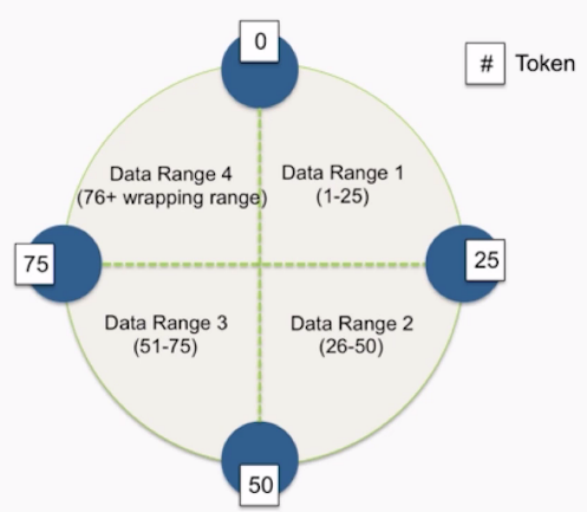

Figura 3: Particionamiento en Cassandra

2.1.2. Cassandra Query Lenguage (CQL)

permite conectarse al cualquier nodo del cluter (homogeneidad)

Apache Cassandra posee su propio lenguaje de consultas, el denominado Cassandra Query Lenguage (CQL). Su sintaxis guarda una gran similitud con la de SQL, lo cual facilita, de forma notoria, el salto que supone pasar de trabajar con bases de datos relacionales a distribuidos.

Aún siendo sintácticamente tan parecido a SQL, presenta ciertas restricciones debido a que es un lenguaje de consultas de una base de datos distribuida. Por ejemplo, no ofrece la posibilidad de realizar operaciones como JOIN y es totalmente necesario especificar todas los atributos que componen la clave primaria a la hora de realizar cualquier consulta de filtrado o de actualización en la tabla. La única operación que no cumple esta restricción es un select que contenga la clausula where, ya que, al definir la clave primaria Cassandra indexa de forma automática todos sus componente, posibilitando más tarde hacer uso de ellos en este caso concreto.

Otra de las peculiaridades que presenta CQL es el hecho de ofrecer dos modos distintos de realizar un update. El primero de todos es el mencionado en el párrafo anterior. El segundo posibilita actualizar una columna realizando un insert repitiendo el valor de las claves primarias de una columna ya existente en la base de datos. Esta segunda forma es cómoda a la par de peligrosa porque Cassandra no notifica si una clave primaria ya existe en la base de datos o no, pudiendo un insert desencadenar en un update no deseado.

2.2. Apache Spark

Apache Spark[9] es un proyecto open source de computación en clúster. Desde el principio fue diseñado para poder ejecutar algoritmos iterativos en memoria sin la necesidad de almacenar en disco los resultados intermedios generados durante el proceso. Esta peculiaridad permite que los procesamientos llevados a cabo con Spark puedan llegar a ser, en algunos casos concretos, 100 veces más rápidos que los de MapReduce[10].

A mediados de 2014, coincidiendo con el lanzamiento de la primera versión, alcanzó la cifra de 465 colaboradores, convirtiéndolo en el proyecto más activo entre los relacionados con el Big Data dentro de la Apache Software Fundation.

Apache Spark está compuesto por múltiples y variados componentes que pueden ser utilizados de forma conjunta.

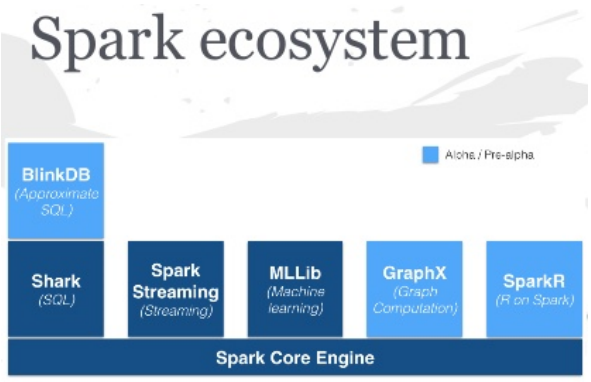

Figura 4: Ecosistema Spark

La base del proyecto es el denominado Spark Core. Proporciona envío distribuido de tareas, planificación y funciones básicas de entrada salida. La abstracción fundamental de programación se llama Resilient Distributed Datasets (RDD)[11], una colección lógica de datos particionados a través de las máquinas que se expone mediante una API integrada en lenguajes como Java, Python y Scala.

2.2.1. Funcionamiento

Para el funcionamiento de Spark, es condición sine qua non que los nodos de la infraestructura tengan acceso a la totalidad de los datos que se desea tratar. Ello implica que para procesar un fichero de 50GB, cada nodo tendría que poseer una copia del mismo

almacenado en su disco. Esta praxis es inviable, ya que más allá de los problemas de consistencia que generaría, para nada es eficiente ocupar la memoria de todos los nodos con información redundante y procesar el fichero entero cuando en realidad se va a hacer uso de una pequeña porción de dichos datos en cada ejecución.

Las bases de datos distribuidas como Cassandra solventan los problemas anteriormente mencionados. Se encargan de distribuir los datos entre diferentes nodos del clúster, ofrecen la posibilidad de acceder a ellos desde cualquier punto y mantienen la consistencia de los mismos a cambio de sufrir una pequeña latencia en el caso de requerir información almacenada en otro nodo de la infraestructura.

Al ejecutar una aplicación que opera con Spark, un componente denominado driver es lanzado. Debido a la necesidad de obtener recursos (CPU y memoria) para llevar a cabo la computación que se le ha encomendado, se comunica con un nodo del clúster que, mediante especificación previa, adopta el rol de maestro. Éste pregunta a todos los nodos que conforman la infraestructura sobre la cantidad de recursos disponibles que poseen y así asignarles los executors correspondiente. Las máquinas que alojen al menos un executor pasan a denominarse worker y a partir de este momento, cada executor podrá comunicarse directamente con el Driver para poder recibir las tareas que éste le envíe.

Un executor es una unidad de trabajo que se encarga de computar las tareas que le encomienda el driver. El número de executors que puede albergar cada worker está directamente relacionado con el número de procesadores que este posee. De la misma forma, es posible repartir la memoria RAM que dispone el nodo worker entre varios executors. Spark permite modificar ambos parámetros programáticamente permitiendo así poder amoldarse a las particularidades de cada ejecución.

Para transferir el código del programa, residente en la máquina del driver, éste adopta el rol de servidor e intenta enviar dicho código a los workers. Si el fichero JAR que contiene el código ha sido recibido correctamente por sus destinatarios, estos responden mediante un ACK y en caso contrario, se vuelve a intentar el envío un número determinado de veces. Una vez llegado al máximo de reintentos, el worker que no haya enviado el ACK es considerado como caído, quedando los executors que albergaba fuera del posterior reparto de tareas.

A la hora de realizar operaciones en Spark, el objeto estrella es el denominado Resilient Distributed Datasets (RDD)[11]. Se trata de una abstracción que mediante diferentes APIs disponibles para Java, Scala y Python permite manipular datos distribuidos por los diferentes nodos del clúster como si estuvieran almacenados de forma local. Este objeto es inmutable, lo cual implica que una vez creado no se le pueden añadir nuevos

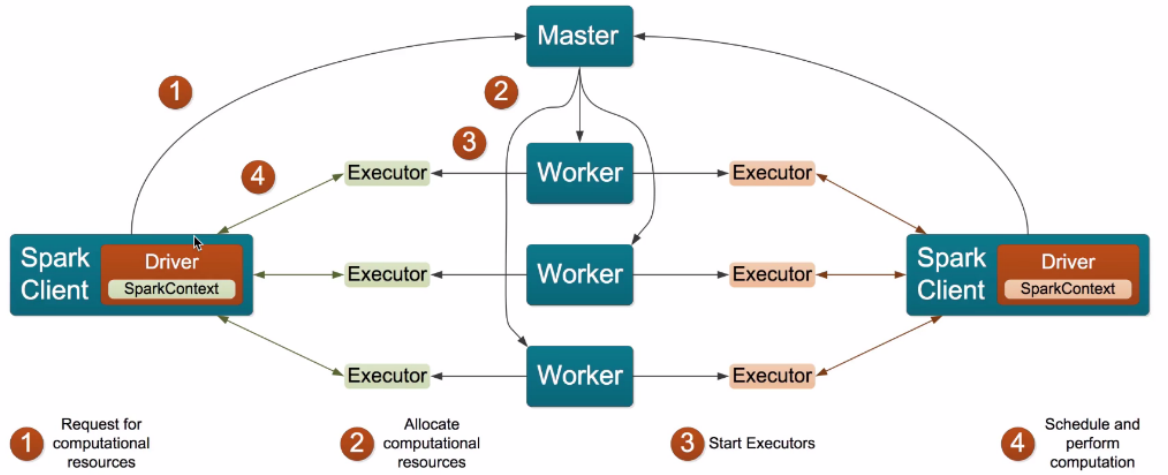

Figura 5: Arquitectura Spark

elementos o eliminar los existentes, solo aplicar transformaciones y acciones sobre el.

Las operaciones que se pueden realizar sobre las RDD se agrupan, tal y como se ha adelantado antes, por transformaciones y acciones. Las primeras transforman un RDD en otro según el criterio indicado y las segundas realizan modificaciones sobre los datos almacenados en dichas RDD. Cabe destacar que las transformaciones en Spark son operaciones "lazy", lo cual implica que en realidad cada nodo memoriza la secuencia de transformaciones que ha de realizar y los procesa cuando una acción es ejecutada.

Una vez terminada la primera fase en la que los executor son creados y enlazados con el driver, éste último empieza a analizar la estructura del código y genera un grafo DAG (Directed Acyclic Graph) con las operaciones que se realizan sobre la RDD. Partiendo de ese grafo genera un job por cada operación de tipo acción que encuentra y dentro de cada job separa la ejecución en diferentes stages según las dependencias que existan entre operaciones. Por último, cada stage es dividido por defecto en unidades de 64MB y a cada unidad resultante se denomina task, el cual es enviado a un executor para ser procesado. El tamaño de cada task puede ser modificado programáticamente, pudiendo de esa forma manipular el número de task que un executor deba ejecutar.

El driver, una vez habiendo recibido los resultados de todas los task que ha repartido, enviará un mensaje a los executors indicando que el procesamiento ha sido finalizado y calculará el resultado final ofreciendo la posibilidad de, por ejemplo, almacenarlo en una base de datos distribuida como Cassandra.

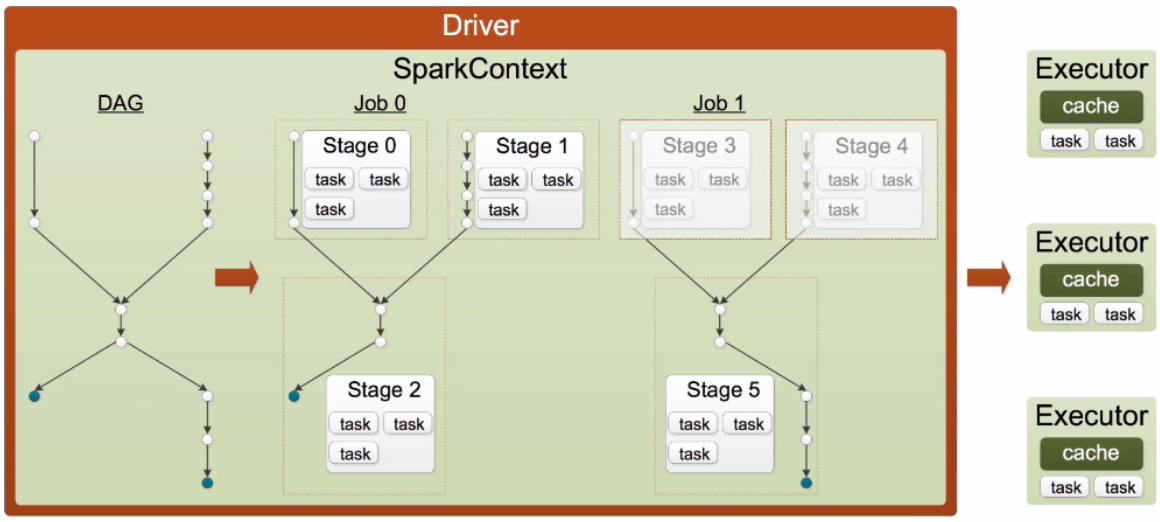

Figura 6: Arquitectura Spark

Bibliografía y Referencias

[1] Fay Chang y col. “Bigtable: A distributed storage system for structured data”. En: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), pág. 4.

[2] Jeffrey Dean y Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. En: *Communications of the ACM* 51.1 (2008), págs. 107-113.

[3] Giuseppe DeCandia y col. “Dynamo: amazon’s highly available key-value store”. En: *ACM SIGOPS Operating Systems Review* 41.6 (2007), págs. 205-220.

[4] Avinash Lakshman y Prashant Malik. “Cassandra: a decentralized structured storage system”. En: *ACM SIGOPS Operating Systems Review* 44.2 (2010), págs. 35-40.

[5] James Manyika y col. “Big data: The next frontier for innovation, competition, and productivity”. En: (2011).

[6] Raghunath Nambiar. “Towards an Industry Standard for Benchmarking Big Data Workloads”. En: ().

[7] Sam Newman. *Building Microservices*. .º’Reilly Media, Inc.", 2015.

[8] Tilmann Rabl y col. “Solving big data challenges for enterprise application performance management”. En: *Proceedings of the VLDB Endowment* 5.12 (2012), págs. 1724-1735.

[9] Alexandru Dan SICOE y col. *A Persistent Back-End for the ATLAS Online Information Service (P-BEAST)*. Inf. téc. ATL-COM-DAQ-2012-010, 2012.

[10] *The State of Big Data in the Large Corporate World*. Big Data Executive Survey 2013 Boston. 2013.

[11] Reynold S Xin y col. “Shark: SQL and rich analytics at scale”. En: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*. ACM. 2013, págs. 13-24.

[12] Matei Zaharia y col. “Spark: cluster computing with working sets.” En: *HotCloud* 10 (2010), págs. 10-10.